BraGuia React Native

March 5, 2024

Diogo Barbosa
PG50326

Tiago Sousa
PG50500

Introdução

Este relatório aborda a fase final do projeto BraGuia, uma aplicação multiplataforma visando servir como um guia turístico interativo para a cidade de Braga. O projeto adotou uma abordagem arquitetural baseada no padrão Flux, tendo sido concebido em React-Native com recurso à plataforma Expo. O Expo foi escolhido principalmente pelas suas poderosas ferramentas de desenvolvimento, inclusão de várias bibliotecas pré-configuradas e a vasta documentação das mesmas.

Durante a implementação, foram consideradas as soluções de persistência de dados, fluxo de dados, gestão do estado e integração com serviços externos, como o Google Maps. O relatório discute as funcionalidades implementadas, as limitações encontradas, bem como a gestão de projeto e controlo de versão utilizados ao longo do desenvolvimento.

Detalhes de Implementação

Estrutura do Projeto

A abordagem arquitetural utilizada foi a Flux, que separa a lógica de dados do UI e centraliza na Store a gestão do estado da aplicação. Na Figura 1 está ilustrada uma representação geral da arquitetura utilizada mais especificamente, tendo em conta que o código foi desenvolvido utilizando React-Native.

As React Views são os componentes de UI que, de acordo com o estado da aplicação, apresentam o conteúdo ao utilizador. Nestes componentes são registadas as interações do utilizador que se traduzirão em actions.

As actions creators são funções que criam actions, que contém o tipo da action e o payload. Estas action são enviadas do UI para o store para fazer a alteração do estado da aplicação por meio de dispatchers.

O store é responsável pela gestão do estado global da aplicação, sendo que é imutável. Mudanças no estado acontecem quando são recebidas actions, que depois de interpretadas, dão origem a um novo estado que é guardado e passado para as React Views.

A estrutura flux providencia um fluxo unidirecional de informação, em que a informação contida no estado flui desde o Store (que gere o estado da aplicação) até às Views (componentes do UI).

O fluxo de informação funciona da seguinte maneira: ao interagir com os componentes do UI o utilizador provoca actions, que são enviadas para o Store, que recebe a action e altera o estado da aplicação conforme; assim que isso acontece, o novo estado é enviado para as Views, que atualizam o conteúdo apresentado se necessário.

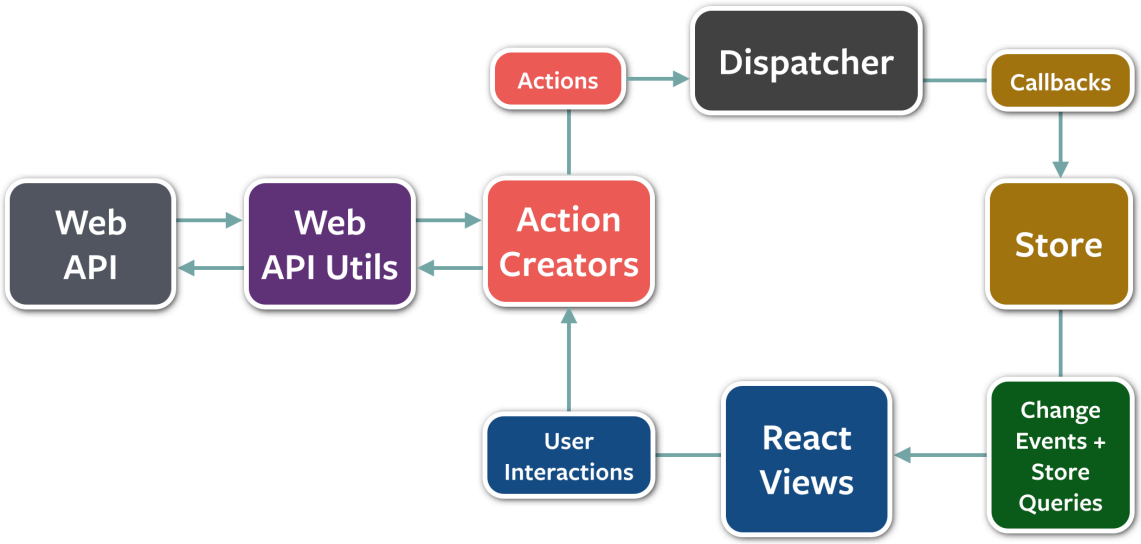

Figura 1: Arquitetura da BraGuia

Soluções de implementação

Navegação

Decidimos utilizar a biblioteca *expo-router* que utiliza uma abordagem baseada em ficheiros para definir e gerir a estrutura de navegação da aplicação.

Alguns exemplos desta abordagem são, por exemplo:

- A rota `/home/trails` executaria o ficheiro `app/home/trails.tsx`
- Para a página de cada Roteiro é usada uma rota dinâmica, em que `/home/trail/1`, executaria o ficheiro `app/home/trail/[id].tsx`.
- Os ficheiros `_layout` são utilizados para definir que tipo de navegação deve ser utilizada, por exemplo, *Stack* ou *Tabs*.

Esta biblioteca mostrou-se bastante útil e ajudou-nos a estabelecer uma boa estrutura do projeto.

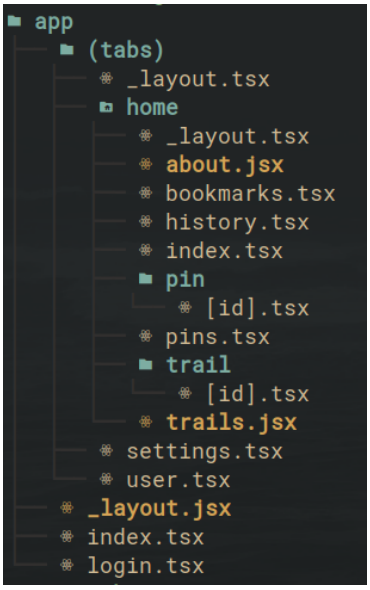

Figura 2: Estrutura de routing

Componentes UI

De modo a agilizar o processo de desenvolvimento do UI usámos o react-native-paper, que contém componentes em conformidade com a biblioteca Material Design.

Os criados foram feitos de forma abstrata de forma a poderem ser reutilizados através de Dependecy Injections.

Gestão do estado Global

Os dados recolhidos da *API*, são diretamente armazenados no estado global da aplicação. Para a gestão do estado da aplicação recorremos à biblioteca redux, que disponibiliza a leitura do estado a todos os componentes de aplicação e um conjunto de ações para poder alterá-lo.

Persistência de dados

Para garantir que a aplicação continua funcional mesmo que não consiga aceder à API devido a problemas de integridade ou de rede, utilizamos a biblioteca redux-persist. Esta biblioteca permite que o estado global da aplicação seja guardado em disco, possibilitando a continuidade do funcionamento da aplicação offline.

As preferências da aplicação foram implementadas ao nível do dispositivo, ou seja, independentemente do utilizador autenticado, as preferências são as mesmas. Foi usada a biblioteca AsyncStorage para guardar localmente as preferências do dispositivo e aplicá-las ao iniciar a aplicação, como, por exemplo, o uso das *Geofences*.

Login/Logout

O login é feito com uma chamada à *API* com o username e password para serem autenticados. Se a resposta for positiva o utilizador entra na aplicação, se não um erro é apresentado. Quando o login é feito com sucesso, a aplicação guarda os cookies que vêm na reposta para poder persistir o login no futuro. No caso do utilizador já ter sido autenticado previamente e não tiver feito logout, a aplicação verifica localmente se existe algum login feito e garante acesso se assim for o caso, sem necessidade de introduzir novamente credenciais.

O logout é feito apagando os cookies guardados no dispositivo. De seguida o utilizador é levado para a página de login, que requer credenciais para iniciar sessão novamente.

Cookies

Os cookies que são recebidos na response ao request de login são guardados localmente através do SecureStore, permitindo persistir o login em futuras utilizações da aplicação. A escolha do SecureStore foi feita tendo em conta que os cookies são uma informação sensível, e por isso devem ser encriptados antes de serem guardados localmente em disco.

Todas as chamadas pós login são feitas com o uso dos cookies armazenados.

Integração com Google Maps

De modo a mostrar um mapa do roteiro e no ponto de interesse, usamos a biblioteca React Native Maps[1]. Para a navegação do roteiro é usado o módulo *Linking* com a *API* de navegação do Google Maps[2].

[1] https://docs.expo.dev/versions/latest/sdk/map-view/

[2] https://developers.google.com/maps/documentation/directions/get-directions#DirectionsRequests

Visualização e Descarregamento de Mídia
Para efetuar a visualização de mídia foi utilizado o componente react <Image/> para as imagens e a biblioteca *expo-av* para áudio e vídeo.

Inicialmente, toda a mídia é fornecida pela API, no entanto, um utilizador premium tem a opção de descarregar e usar mídia local. Para isso foi usado o *expo-file-system* para poder fazer o download da mídia e implementado um sistema de cache para dar prioridade aos dados locais, evitando fazer requests redundantes. Desta forma, sempre que o utilizador consulta uma página é verificado primeiro se a mídia existe localmente, e só se não existir é que a aplicação usa a *API*.

Localização e Notificações de Proximidade
Para isto ser possível começamos por pedir ao utilizador pelas permissões necessárias de localização. De seguida, caso o utilizador tenha permitido, o serviço de Geofencing, da biblioteca expo-location é iniciado, de forma a monitorizar todos os pontos de interesse com um raio de 150 metros. Este serviço é executado em *background* com o uso da biblioteca expo-task-manager, definido a partir de uma Task.

Apesar de não termos conseguido implementar com sucesso as notificações, quando o utilizador entra numa *Geofence* seria enviada uma notificação, com o uso da biblioteca *expo-notifications* com o *id* do ponto de interesse.

De modo a provar que o serviço está funcional incluímos as seguinte Figura 3.

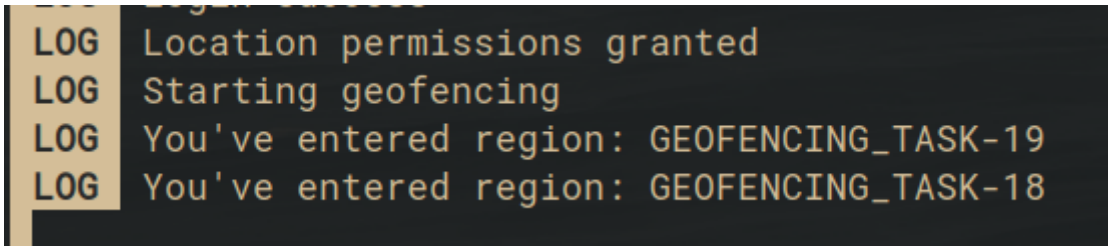

Figura 3: Log após a entrada em duas *Geofences*.

Bibliotecas/dependências utilizadas

- react-redux (https://react-redux.js.org/)
- react-persist (https://redux-toolkit.js.org/rtk-query/usage/persistence-and-rehydration)
- expo-linking (https://docs.expo.dev/versions/latest/sdk/linking/)
- expo-location (https://docs.expo.dev/versions/latest/sdk/location/)
- expo-notifications (https://docs.expo.dev/versions/latest/sdk/notifications/)
- react-native-async-storage (https://react-native-async-storage.github.io/async-storage/docs/usage)
- react-native-paper (https://reactnativepaper.com/)
- expo-file-system (https://docs.expo.dev/versions/latest/sdk/filesystem/)
- expo-av (https://docs.expo.dev/versions/latest/sdk/av/)
- expo-font (https://docs.expo.dev/versions/latest/sdk/font/)
- expo-router (https://docs.expo.dev/router/introduction/)
- expo-secure-store (https://docs.expo.dev/versions/latest/sdk/securestore/)
- expo-task-manager (https://docs.expo.dev/versions/latest/sdk/taskmanager/)

Padrões de software utilizados

Dependency Injection: utilizada em componentes tal como o *card* que mostra a preview de um trail, permitindo que este componente possa ser reutilizado em várias páginas (histórico, bookmarks e trails), adaptando-se ao contexto

Flux: utilizado a gestão do estado global da aplicação e a garantia dos princípios de SSOT (Single Source of Truth) e de Unidirectional Data Flow

Provider pattern: utilizado para tornar o estado da aplicação global, accessível a partir de qualquer componente

Mapa de Navegação

Figura 4: Mapa de navegação

Funcionalidades

Funcionalidades implementadas

- A aplicação deve possuir uma página inicial onde apresenta as principais funcionalidades do guia turístico, descrição, etc.
- A aplicação deve mostrar num ecrã, de forma responsiva, uma lista de roteiros disponíveis;
- A aplicação deve permitir efetuar autenticação;
- A aplicação deve possuir a capacidade de efetuar chamadas para contactos de emergência da aplicação através de um elemento gráfico facilmente acessível na aplicação.
- A aplicação deve assumir que o utilizador tem o Google Maps instalado no seu dispositivo (e notificar o utilizador que este software é necessário);
- A aplicação deve suportar 2 tipos de utilizadores: utilizadores standard e utilizadores premium;
- A aplicação deve guardar (localmente) o histórico de roteiros e pontos de interesse visitados pelo utilizador;
- A navegação proporcionada pelo Google Maps deve poder ser feita de forma visual e com auxílio de voz, de modo a que possa ser utilizada por condutores;
- A aplicação deve ter a capacidade de apresentar e produzir 3 tipos de mídia: voz, imagem e vídeo;
- A aplicação deve possuir uma página de informações acerca do utilizador atualmente autenticado;
- A aplicação deve possuir um menu com definições que o utilizador pode manipular;
- A aplicação deve possuir uma página que mostre toda a informação disponível relativa a um ponto de interesse: localização, galeria, mídia, descrição, propriedades, etc;
- A aplicação deve mostrar, numa única página, informação acerca de um determinado roteiro: galeria de imagens, descrição, mapa do itinerário com pontos de interesse e informações sobre a mídia disponível para os seus pontos;
- A aplicação deve possuir a capacidade de iniciar um roteiro;
- A aplicação deve possuir a capacidade de ligar, desligar e configurar os serviços de localização;
- Para utilizadores premium (e apenas para estes) a aplicação deve possibilitar a capacidade de navegação, de consulta e descarregamento de mídia;
- A aplicação deve possuir a capacidade de descarregar mídia do backend e aloja-la localmente, de modo a poder ser usada em contextos de conectividade reduzida;

Funcionalidades não implementadas

- A aplicação deve possuir a capacidade de interromper um roteiro;
- A notificação emitida quando o utilizador passa pelo ponto de interesse deve conter um atalho para o ecrã principal do ponto de interesse;

Funcionalidades parcialmente implementadas

- A aplicação deve possuir a capacidade de emitir uma notificação quando o utilizador passa perto de um ponto de interesse;

Discussão de resultados

Trabalho realizado

Apesar de não termos conseguido implementar todos os requisitos, consideramos que o produto final foi bem concebido, não faltando muito para atingir tudo o que foi pedido. Além disso, o grupo também considera que deveria ter explorado a vertente de testes unitários no código produzido, especialmente tendo em conta a grande complexidade e dimensão do projeto.

Limitações

1. Na implementação do requisito funcional das notificações de proximidade de um ponto de interesse, não conseguimos implementar na totalidade a emissão das notificação. Isso resultou em não conseguirmos implementar a navegação através da notificação para o ponto de interesse.

   Obs: apesar de o código ter sido parcialmente implementado, decidimos mantê-lo no repositório, mas não o utilizar na solução final.
2. Na implementação do *logout*, quando executamos a chamada à *API* com o `POST` *logout*, tendo os cookies do utilizador logged in, recebia o código de erro 403 como resposta, e em posteriores chamadas o código inicialmente esperado, 200.
3. O *dark theme* não é persistente, o *theme* selecionado ao iniciar a aplicação é sempre o do dispositivo, só a partir daí é que pode ser alterado nas definições da aplicação
4. Os dados pertencentes ao utilizador (bookmarks e histórico) não são persistentes entre mudanças de utilizador.

Funcionalidades extra

Implementamos funcionalidades extras, como os Bookmarks, onde um utilizador pode marcar um roteiro específico, que será então guardado numa lista própria para que o utilizador possa aceder facilmente aos seus roteiros favoritos sempre que desejar. Além disso, introduzimos a opção para o utilizador escolher entre *light* e *dark theme*, permitindo ao utilizador adaptar a interface visual conforme as suas preferências pessoais ou condições de iluminação ambiente.

Gestão de projeto

Gestão e Distribuição de trabalho

- Diogo
  - Estado global
  - Device Preferences
- Tiago
  - Integração com Google maps
  - Geofences
  - Media Player
  - Networking
- Ambos
  - UI
  - Navegação

Eventuais metodologias de controlo de versão utilizadas

Para o controlo de versões foi usado o GitHub, permitindo que os membros do grupo trabalhassem em paralelo no código em funcionalidades diferentes, fazendo *merge* e resolvendo eventuais conflitos.

Conclusão

O projeto BraGuia proporcionou uma valiosa oportunidade de aprendizagem em diversos aspetos do desenvolvimento de aplicações móveis multiplataforma. Ao longo deste processo, enfrentamos desafios que nos permitiram adquirir novos conhecimentos e habilidades, ao mesmo tempo, em que reforçamos conceitos previamente aprendidos.

O projeto apresenta algumas limitações tal como a ausência de notificações de proximidade e a persistência total dos dados da aplicação. Apesar dessas dificuldades estamos confiantes que o trabalho ficou bem conseguido, com quase todos os requisitos funcionais implementados com sucesso e ainda algumas funcionalidades extra.

No geral, o projeto BraGuia não só nos permitiu aplicar os conhecimentos teóricos adquiridos em sala de aula, como nos desafiou a expandir as nossas habilidades técnicas, a trabalhar em equipa de forma eficaz e a enfrentar problemas complexos com determinação e criatividade.